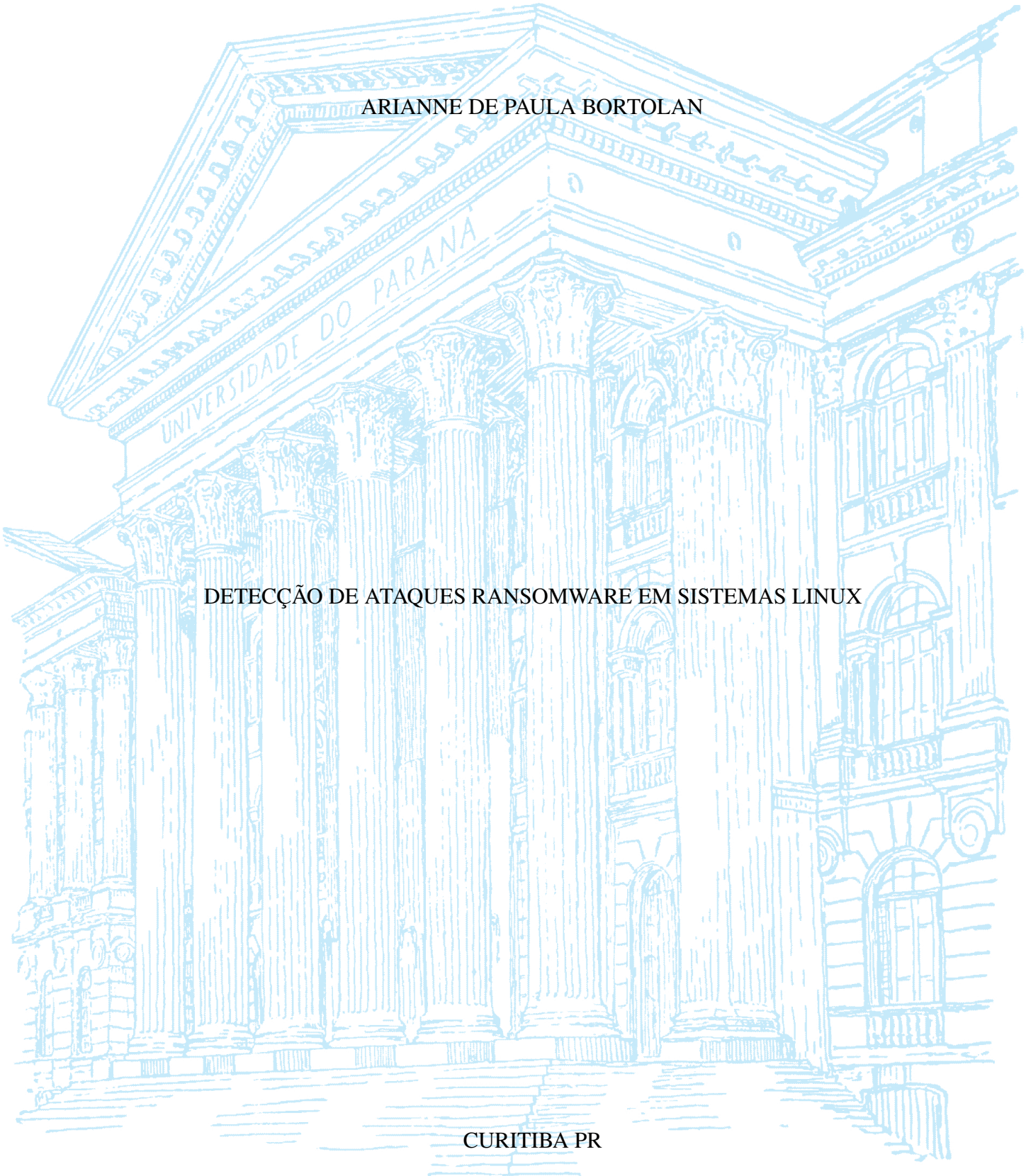
UNIVERSIDADE FEDERAL DO PARANÁ

ARIANNE DE PAULA BORTOLAN



DETECÇÃO DE ATAQUES RANSOMWARE EM SISTEMAS LINUX

CURITIBA PR

2019

ARIANNE DE PAULA BORTOLAN

DETECÇÃO DE ATAQUES RANSOMWARE EM SISTEMAS LINUX

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

CURITIBA PR

2019

Ficha catalográfica

Substituir o arquivo 0-iniciais/catalografica.pdf (PDF em formato A4) pela ficha catalográfica fornecida pela Biblioteca da UFPR a pedido da secretaria do PPGInf/ UFPR.

O conteúdo exato da ficha catalográfica é preparado pela Biblioteca Central da UFPR, a pedido da secretaria do PPGINF. Portanto, não "invente" um conteúdo para ela.

ATENÇÃO: por exigência da Biblioteca da UFPR, esta ficha deve ficar no verso da folha de rosto (que contém o nome do orientador e área de concentração). Cuide desse detalhe quando imprimir as cópias finais.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

Dedico esse trabalho a todos que me ajudaram e apoiaram ao longo dessa jornada, e também a todos que dúvidaram dela, suas dúvidas me motivaram a nunca desistir.

AGRADECIMENTOS

Deixo aqui meu agradecimento ao meu orientador e sua incrível paciência apesar de todos meus sumiços e procrastinações, sua paciência e conhecimento me ajudaram muito a concluir esse trabalho. Também agradeço aos meus amigos (bonde) que sempre estiveram ao meu lado e me apoiaram nos momentos em que mais quis desistir dessa graduação (e com isso me impediram de desistir da mesma), vocês são minha segunda família e sou eternamente grata a tudo. Agradeço também a minha namorada Lara, por estar do meu lado em um ano tão caótico e me trazer paz e por último e não menos importante agradeço a minha mãe, por ter me apoiado desde criança a sempre lutar pelos meus sonhos, graças a isso estou aqui escrevendo esse agradecimento.

RESUMO

Um ransomware é um software malicioso que cifra arquivos de usuários e pede um resgate em troca. Após o famoso ataque do ransomware Wannacry, esse tipo de malware se popularizou causando problemas em grande escala e perda de informações e de dinheiro. O objetivo desse trabalho de graduação é apresentar um modelo de detecção que identifica ataques da ransomwares em tempo real. O algoritmo projetado para esse modelo se baseia na mudança de estado dos arquivos, ele compara o estado de um arquivo após ser escrito com seu estado anterior, caso este exista. A comparação é feita baseada em três métricas: entropia de Shannon, Fuzzy Hashes e Magic numbers do arquivo. Para conseguir analisar arquivos após a sua escrita é necessário que sejam interceptadas suas chamadas de escrita. Por esse motivo foi implementado um sistema de arquivos Fuse escrito em Python utilizando a biblioteca Fusepy, tanto o sistema de arquivos Fuse quanto o algoritmo proposto foram escritos em Python 3. O algoritmo proposto para comparação consegue identificar que arquivos foram cifrados, esse resultado foi obtido em testes unitários somente com o algoritmo, sem integrá-lo ao Fuse. Já quando inserido no sistema de arquivos o algoritmo simplesmente não funcionou, causando quebras nas chamadas de escrita do sistema de arquivos. Algumas das causas desse problema se devem a biblioteca Fusepy em si, visto que ela foi descontinuada e que muitas funções na chamada de escrita são assíncronas, impedindo o algoritmo de detecção de executar conforme planejado.

Palavras-chave: Ransomware. Linux. Fuse.

ABSTRACT

Ransomware is a malicious software that ciphers user files and asks for a ransom as a exchange for the user to get his files back. After the famous ransomware attack made by the malware Wannacry, ransomwares became a popular malware used by criminals, resulting in large impact attacks where both private and public institutions lose lots of data and money. The main objective of this article is to present a model of ransomware detector that can identify a ransomware attack in realtime. The proposed algorithm analise changes in a file state after is written, the actual state after the write is compared with the state before the write if there is any. Three metrics are used for comparison, Shannon Entropy, Fuzzy Hashing and Magic numbers. For the algorithm to be able to make the analisys it needs to be executed after a write system call, for this reason it was implemented a Fuse filesystem so the detection algorithm could work together with a filesystem intercept the syscall. Both Fuse filesystem and algorithm use Python 3, the Fuse filesystem uses the library Fusepy. In unitary experiments the algorithm was able to detect changes in files that were ciphered but when executed inside the Fuse syscalls, the algorithm breaks the filesystem, not being able do to any detection. The reason for such problem is that the Fusepy library is deprecated and the write syscalls work in a asynchronous way, making impossible to the detection algorithm to work.

Keywords: Ransomware. Linux. Fuse.

SUMÁRIO

1 INTRODUÇÃO

Um *ransomware* é um tipo de software malicioso (malware) cujo objetivo é a cifragem dos arquivos de um usuário, para então pedir uma quantia de dinheiro em troca do "resgate" (decifração) desses arquivos. Após o ataque em escala mundial do *ransomware* WannaCry, *ransomware* se popularizou entre criminosos e constantemente instituições publicas e privadas sofrem ataques desse tipo de *malware* , resultando em perda de dados e grandes quantias em dinheiro.

Como *ransomwares* têm a característica de ação rápida e altamente destrutiva, os softwares utilizados para detecção de anomalias de softwares maliciosos em um sistema, como o antivírus, não conseguem identificar a invasão a tempo de conseguir impedir a ação do ransomware. Isso ocorre porque, para conseguir detectar um ataque real, é necessário que o antivírus ou softwares utilizados para proteção identifiquem que os arquivos do usuário já foram efetivamente cifrados pelo *malware* . Algumas soluções para esse problema começaram a ser desenvolvidas para sistemas operacionais *Windows*, sendo algumas delas o *CryptoDrop* (Scaife et al., 2016), ShieldFS (Continella et al., 2016) e Redemption (Kharraz e Kirda, 2017). As três soluções usam métricas semelhantes para identificar ataques, tanto o ShieldFS (Continella et al., 2016) quanto o Redemption usam uma tabela de pontos e monitoramento do comportamento de processos, enquanto o *CryptoDrop* (Scaife et al., 2016) foca principalmente na mudança do estado de arquivos.

Apesar das propostas feitas, *ransomwares* continuando causando problemas em larga escala (Cimpanu, 2019a; Lima, 2019) e lentamente novos *ransomwares* para sistemas *Linux* vêm surgindo (Cimpanu, 2019b) mas estes não tem recebido propostas de soluções contra ataques desse tipo.

O objetivo desse trabalho é oferecer um modelo de algoritmo detector de ransomwares, que possa ser aplicado de maneira nativa em sistemas *Linux*. A ideia é que o algoritmo seja executado após as chamadas de escrita do sistema de arquivos, e assim verifique a mudança de estado de cada arquivo que é escrito, a fim de monitorar mudanças e detectar se o arquivo foi cifrado.

O modelo de algoritmo detector apresentado nesse trabalho, por motivos de conveniência será chamado *LookRansom*. Apesar do algoritmo em si ter obtido resultados interessantes em comparações de arquivos cifrados com suas versões originais, o *LookRansom* não é um modelo completo, e deve ser observado como um protótipo de um modelo conceitual de detector nativo em sistemas *Linux*.

A ideia do *LookRansom* é inserir o seu algoritmo de detecção e comparação nas chamadas de sistema de escrita de arquivos. Essa modificação funciona de maneira que o detector monitore alterações nos estados de arquivos que foram escritos ou criados recentemente, fornecendo assim uma defesa nativa ao sistema contra ransomwares.

Apesar de propostas semelhantes já terem sido feitas para sistemas operacionais *Windows* com o ShieldFS por exemplo, como mencionado, ainda existe uma defasagem em sistemas *Linux* contra esse tipo de *malware* , o que diferencia a proposta desse trabalho com a solução do ShieldFS. Além disso, o *LookRansom* se baseia puramente na comparação de estado de arquivos antes e depois de sua escrita, simplificando em muito o algoritmo de monitoramento quanto comparado com as soluções anteriormente mencionadas.

Para manter o escopo deste trabalho simples, optou-se por utilizar o sistema de arquivos Fuse (Wikipedia, 2017) para a implementação em conjunto do *LookRansom*. A razão é devido

à dificuldade que se encontra na implementação de um sistema de arquivos em si (Bertozzi, 2011). Para realizar os testes do algoritmo seria necessário alterar em nível de *kernel* o sistema de arquivos, algo que é trabalhoso e resulta em muitos problemas como o conhecido *kernel panic* (Bertozzi, 2011). Como não era de interesse incluir no escopo deste trabalho esse processo complexo e trabalhoso, ainda mais considerando o tempo disponível para o desenvolvimento do trabalho, escrita e testes, optou-se pelo uso do Fuse, visto que ele oferece acesso a chamadas de sistema de maneira direta, sem todos os detalhes e riscos de uma implementação em *kernel*.

O conteúdo deste trabalho está divido em três partes os capítulos 1 e 2 contêm a introdução e apresenta o estado da arte de *ransomwares* e as soluções encontradas atualmente com relação a ataques; O capítulo 3 apresenta o *CryptoDrop*, um software comercial para *Windows* que serviu de inspiração para a criação do modelo conceitual do *LookRansom*. Além dele são apresentados detalhes da implementação e a proposta deste trabalho de graduação em si; No capítulo 4 são apresentados os experimentos feitos com o algoritmo e sistema de arquivos, dificuldades encontradas e os resultados, e por fim o capítulo 5 contêm a conclusão e reflexões sobre os resultados do capítulo 4.

2 *RANSOMWARE* E ESTADO DA ARTE

Para lidar com um problema é necessário entender sua origem e causas, esse capítulo visa introduzir e explicar o que é um ransomware, seus tipos, vetores de ataque e apresentar algumas famílias e soluções que atualmente são utilizadas para resolver esse problema.

2.1 CONCEITO E HISTÓRIA

Um *ransomware* é um software malicioso que tem como objetivo bloquear o acesso da vítima aos seus arquivos, e então pedir uma quantia de dinheiro em troca do retorno dos acessos. Embora os primeiros *ransomwares* tenham surgido em meados de 1980 (Kharraz et al., 2016), a popularização desse tipo de *malwares* começou nos anos 2000.

Um dos fatores do ganho de popularidade tardio se deve aos riscos envolvendo o pedido de quantias de dinheiro e forma em que elas seriam recebidas pelo atacante. Apesar dos riscos iniciais, os primeiros ataques de *ransomware* se iniciaram propriamente dito no ano de 1989 com o PC Cyborg-Trojan (também conhecido como AIDS) (Wikipedia, 2008), o ransonware visava atacar instituições de saúde. O PC Cyborg-Trojan utilizava de engenharia social como método de propagação, as vítimas recebiam um disquete com supostas informações sobre a crise de AIDS que ocorria na época, e inseriam o disco em suas máquinas. Após a instalação do ransomware, as máquinas ainda tinham que ser reinicializadas 90 vezes antes do *malwares* efetivamente agir, quando era ativado, o *ransomware* efetivamente cifrava os arquivos da máquina e um pedido de resgate aparecia na tela do usuário, informando para que ele enviasse uma quantia em dinheiro para uma caixa postal no Panamá (Wikipedia, 2008).

Combinando as cripto-moedas com a popularização da internet anos depois, os ataques mudaram e se tornaram mais simples, em vez de dependerem da engenharia social e acesso físico a máquina com um disquete, agora esses ataques podem ser remotos e a anonimidade do atacante pode ser garantida. As chamadas cripto-moedas, como o Bitcoin por exemplo, foi o que possibilitou o uso efetivo do *ransomware* por criminosos. Sendo as cripto-moedas, um dos elementos que impulsionou o que foi chamado de "Boom" dos *ransomwares* em meados de 2016 a 2017 (Baker, 2017). Por realizarem transações supostamente anônimas e impossíveis de rastreio, moedas como o Bitcoin, permitem que atacantes mantenham sua anonimidade durante o pedido de resgaste, e assim tornando os ataques seguros e lucrativo ao minimizar o risco do atacante ser localizado.

Em 12 de Maio de 2017 um dos maiores ataques utilizando um *ransomware* aconteceu, esse *ransomware* ficou conhecido mais tarde como WannaCry. O WannaCry é um tipo de *ransomware* que se propaga como um *worm* [1] (verme em português) (Reis, 2017), utilizando o *exploit EternalBlue* (Security, 2017), esse *exploit* se aproveita de uma vulnerabilidade encontrada no sistema operacional *Windows* na implementação do protocolo SMBv1,*Server Message Block*, para se propagar em máquinas de uma mesma organização e invadi-las. Foi estimado que pelo menos 150 países foram afetados pelo ataque do WannaCry nesse dia, infectando milhares de máquinas e causando diversos problemas como a parada de hospitais e outros serviços públicos e privados. O ataque afetou principalmente instituições públicas que não realizaram a atualização de seus sistemas operacionais, visto que a vulnerabilidade explorada pelo WannaCry (*EternalBlue*), tinha sido descoberta em abril de 2017 e correções de segurança já haviam sido lançadas para solucionar essa vulnerabilidade (Palmer, 2017) (Security, 2017).

[1] Software malicioso que o objetivo é infectar o maior número de máquinas possíveis

Outro ataque de *ransomware* que também ocorreu em 2017 foi o NotPetya (uma variação do *ransomware* Petya), ele também utilizou o *EternalBlue* como um dos métodos de propagação. O vetor inicial de ataque era via e-mail, utilizando engenharia social para que as vítimas baixassem o *malwares* e o executassem ao abrir o arquivo malicioso com o NotPetya, e então o *ransomware* durante sua execução utilizava o *EternalBlue* e outras ferramentas que exploravam vulnerabilidades no protocolo SMB. Diferente do WannaCry em que seu plano de ataque era simplesmente se espalhar para o maior número de máquinas possíveis, o NotPetya visava atacar diretamente organizações localizadas na Ucrânia (Greenberg, 2018).

É interessante mencionar que o *exploit EternalBlue*, mencionado nos dois ataques anteriores, ainda é comumente utilizado (Scammell, 2019) em outros *ransomwares* e *malware* s de outros tipos como um vetor de infecção. Apesar de não entrar no escopo o seu funcionamento e efeitos, ele serve como exemplo de que muitas vezes certas ferramentas de *exploit* podem ser utilizadas por mais de um tipo de *malwares* para diferentes tipos de ataque.

Também é importante lembrar que, embora o WannaCry e NotPetya sejam *malware* s que usam uma campanha de ataque similar à de um *worm*, isso não significa que esse seja o único tipo de vetor de ataque utilizado pelos *ransomwares*. Uma tática ainda muito popular é a de utilização de e-mails de contendo os softwares maliciosos e mensagens com uma certa engenharia social que conseguem convencer as vítimas a executar esses programas em seus computadores.

Além de ataques de grande impacto midiático como o WannaCry e NotPetya, constantemente são lançadas noticias sobre ataques de pequeno a médio porte de ransomware; O site de noticias especializado em segurança da informação, ZDnet, estimou que em 2019 um total de 500 escolas foram vítimas de ataques de *ransomware* nos Estados Unidos da América (Cimpanu, 2019a). Apesar de não estar mais em seu momento de popularidade, como indica um relatório da *malware* Bytes, *ransomwares* ainda são um problema, principalmente para organizações publicas e privadas que se tornaram o principal alvo dos atacantes que utilizam esse tipo de software, já que visam um maior lucro (MalwarebytesLabs, 2019).

O relatório técnico de 2019 da Sophos (Andrew Brandt, 2019) mostrou que recentemente os ataques de *ransomware* passaram a ser feitos em alvos específicos, nesse caso empresas privadas. Também foi mostrado nesse relatório, que estima-se que os desenvolvedores do *ransomware* SamSam, conseguiram desde 2016 um rendimento em seus ataques de um total de 6.5 milhões de dólares, em média isso seria um lucro maior que 100 mil dólares por ano (Andrew Brandt, 2019). Esses números não somente demonstram o prejuízo que *ransomwares* têm causado para empresas nos últimos 3 anos, mas também a necessidade de soluções eficazes para casos de ataques feitos com sucesso, visto que cada vez mais esse *malwares* tem sido utilizado para ataques organizados e com vítimas especificas.

2.2 TIPOS DE RANSOMWARE

A família de *ransomwares* é um tanto extensa, como mostra um relatório técnico da SophosLabs em 2015 (James Wyke, 2015). Apesar disso, pode-se de uma maneira simplista, dividir os *ransomwares* em dois grupos genéricos(Al-rimy et al., 2018)(Kharraz et al., 2015): *File-encrypting* e *lockers*.

Ransomwares do tipo *File-encrypting*, em uma tradução direta para o português: cifrador de arquivos, tem o príncipio de cifrar, ou realizar outras tarefas destrutivas como remoção de arquivos, tornando esses arquivos inutilizáveis e impedindo que a vítima do ataque consiga recuperá-los, a menos que utilize algum tipo de cópia externa. Esse tipo de *ransomware* é o mais conhecido na mídia, visto que um dos ataques de *malwares* mais famosos utilizando ransomware,

foi o WannaCry em 2017. O WannaCry (Al-rimy et al., 2018) tem um comportamento de ataque semelhante a um típico *file-encrypting*, embora para vetor de ataque ele funcionasse como um *malwares* do tipo *worm*. Após infectar a máquina do usuário, o WannaCry cifrava os arquivos e então pedia uma quantia de dinheiro em troca da suposta chave com que os arquivos poderiam ser decifrados e retornados ao seu estado original.

Os *ransomwares* que se classificam como *lockers*, traduzindo para o português: bloqueador, não tem o mesmo procedimento destrutivo que os *file-encrypting*. Em vez disso, eles bloqueiam o acesso ao sistema, inserindo uma tela semelhante a um bloqueador de tela normal, só que somente se o usuário pagar a quantia de resgate, ele então conseguirá realizar seu acesso no sistema e acessar seus arquivos. Esse tipo de *ransomware* é considerado menos destrutivo, pois em muitos casos ele não modifica os arquivos, sendo possível o usuário recuperá-los de outras maneiras sem pagar a quantia pedida pelo atacante. Apesar dessas definições, os *ransomwares* podem acabar não se prendendo a elas e muitas vezes misturar os tipos de ataque, podendo alguns *lockers* também realizarem a cifragem de arquivos, ou um *file-encrypting* algum tipo de bloqueio de tela para que o usuário não consiga sobrepor nenhuma tela sobre a mensagem do ataque pedindo o montante de dinheiro.

De acordo com (Kharraz et al., 2015), apesar da variedade de métodos utilizados por esses *ransomwares* e a melhora na utilização de técnicas de encriptação e destruição de dados, os ataques ainda são superficiais e feitos somente realizando algum tipo de bloqueio do usuário do sistema, ou cifrando somente alguns arquivos de sua máquina, sem muita complexidade.

Além dessa divisão simplista, os *ransomwares* também são divididos entre famílias, o termo família quando se trata de *malware* s é utilizado para descrever um tipo de especifico de *malwares* e seu ciclo de vida e variações. Nas próximas subseções serão mostradas algumas famílias de *ransomware* e um resumo de suas ações de ataque. Essas não são todas as famílias existentes de *ransomwares* e sim apenas uma pequena amostra para uma análise breve, em que possamos perceber e comentar sobre os padrões e diferenças encontradas nas variações de *ransomwares*.

2.2.1 CryptoWall

Provavelmente o *ransomware* mais antigo dessa lista, sendo suas primeiras infecções datadas de 2014, o CryptoWall (James Wyke, 2015) infecta suas vítimas através de spam via e-mails e injeção de código em propagandas enganosas em sites ou via outros *malware* s (Petters, 2018). Uma vez executado na máquina de sua vítima, o CryptoWall cria uma lista de registros com a lista de todos os arquivos a serem cifrados, porque caso os arquivos sejam recuperados e o computador reiniciado, o CryptoWall cifra novamente esses arquivos, além disso ele não somente funciona como um *ransomware* que cifra seus arquivos e pede resgate, ele também funciona como um trojan e tenta capturar senhas do usuário infectado. Ele usa o algoritmo RSA-2048 para realizar a cifragem de arquivos (Petters, 2018).

2.2.2 Ryuk

O mais recente *ransomware* desta lista, o Ryuk tem tido uma campanha de infecção em grande escala no ano de 2019, tendo como principal vetor de ataque campanhas de spam com softwares maliciosos que o Ryuk usa de vetor, sendo um deles o Trojan Bancário Emonet. O Ryuk tem como foco atacar organizações privadas, os criminosos por trás do Ryuk, um grupo focado em crimes virtuais auto denominado *WIZARD SPIDER*, altera o *malware* de acordo com a vítima em mente, sendo seu ataque feito de maneira direta e intencionada (Hanel, 2019). O *malware* tem técnicas de evasão para evitar ser analisado ou ativado em vítimas que não sejam de

interesse. Por exemplo, o *malwares* verifica o sistema em que está rodando e caso esteja em uma máquina virtual ou a linguagem definida seja Russo, o Ryuk não é ativado. O *malware* utiliza algoritmos de cifragem RSA-2048 e AES-256 para realizar a cifragem de arquivos, além disso ele cifra não somente arquivos em disco local, mas também vasculha à rede a busca de cópias para realizar a cifragem delas também.

Um caso curioso do Ryuk é que, diferente de muitos *ransomwares* que têm sua lista branca com extensões de arquivos que não devem ser cifradas, como arquivos cruciais para o funcionamento de sistemas, o Ryuk tem uma lista em que somente não cifra arquivos executáveis e bibliotecas dinâmicas, fazendo com que o usuário possa acabar não somente perdendo seus arquivos e cópias, como também causar danos ao sistema operacional em si.

2.2.3 TeslaCrypt

A família de *ransomwares* TeslaCrypt (James Wyke, 2015), também conhecida como EccKrypt, tem o comportamento um *file-encrypting* ransomware, o TeslaCrypt realiza a cifragem de arquivos de usuário e então pede uma quantia de dinheiro em resgate para a decifração desses arquivos. Ele utiliza o algoritmo de cifragem simétrica AES. Seu principal vetor de ataque é o *exploit* Angler. O *exploit* em si trata-se de uma injeção de código no iframe do site que contenha a vulnerabilidade em seu *Adobe Flash* e então é realizada uma redireção de página. O *exploit* é de alta complexidade, visto que utiliza técnicas de evasão de máquinas virtuais e de analisadores de *malwares* durante sua execução. Uma vez feita as execuções de evasão e percebido ser um usuário vulnerável legitimo, então é feito o download do *ransomware* propriamente dito na máquina. Então, ao ser executado o TeslaCrypt inicia alguns dos seguintes processos em *threads* diferentes:

- Cifragem dos arquivos do usuário;
- Monitoramento e encerramento dos processos: *cmd.exe*, *msconfig*, *regedit*, *procexp*, *taskmgr*;
- Exclusão de todos os volumes de backup utilizando o *vssadmin.exe*.

Apesar da complexidade do ataque, o TeslaCrypt já tem sua assinatura reconhecida por alguns antivírus (Wikipedia, 2015) e também não é considerado funcional, visto que a chave que decifra os arquivos encriptados pelo TeslaCrypt foi divulgada oficialmente pelos seus desenvolvedores, tornando possível as vítimas do *ransomware* recuperarem seus arquivos cifrados.

Curiosamente, o TeslaCrypt tinha como foco atacar usuários que costumassem utilizar seus computadores para jogar (Wikipedia, 2015), visto que o *malwares* focava na cifragem de arquivos em que as extensões fossem de jogos.

2.2.4 CTB-Locker

Assim como o TeslaCrypt, o CTB-Locker (James Wyke, 2015) é um *File-encrypting* ransomware, ele cifra os arquivos do usuário e então pede uma quantia em resgate pela decifração desses arquivos. O CTB-Locker utiliza como principal vetor de ataque, campanhas de spam e softwares terceiros que carreguem como *payload* o CTB-Locker. Os *ransomwares* da família CTB-Locker utilizam para a cifragem dos arquivos uma mistura dos algoritmos AES e ECC (Cryptografia por curva elíptica). A cifragem dos arquivos do usuário em si é feita utilizando o AES, e então a chave que realiza a decifração desses arquivos é cifrada utilizando o ECC. O

CTB-Locker tem como foco cifrar somente arquivos com extensões específicas como arquivos `.txt`, `.doc`, imagens etc. Além disso o *malware* também tem duas variações, uma que foca em atacar usuários comum e outra que foca no ataque em servidores de empresas.

2.2.5 Lilocked

Apesar de já ter infectado um total de 6000 servidores de páginas Web, o *ransomware* Lilocked ainda não teve uma amostra capturada para analise (Abrams, 2019). Sendo as informações apresentadas aqui superficiais e baseadas na observação dos ataques ocorridos. Uma das vítimas relatou que suspeitava que Lilocked invadiu seu servidor utilizando uma vulnerabilidade no programa *Exim*, já que esse se encontrava desatualizado (Abrams, 2019). Enquanto outra vítima relatou uma suspeita era que o *ransomware* tinha se aproveitado de alguma vulnerabilidade no seu *Wordpress*, já que este também se encontrava desatualizado (Abrams, 2019). Com essas informações é possível observar que ainda não se sabe muito com relação ao vetor de ataque do Lilocked. O pouco de informações coletadas são que ele não cifra arquivos de sistema, que utiliza os algoritmos de cifragem AES e RSA e que até o momento as principais vitimas são servidores de páginas Web que utilizam sistemas *Linux*.

A Tabela 2.1 apresenta a lista de *ransomwares* mencionados neste trabalho e suas características de maneira sucinta.

ransomware	Sistema Operacional	Tipo	Algoritmo de Cifragem	Vetor de Ataque/ Vulnerabilidade
WannaCry	*Windows*	*file-encrypting* e *locker*	RSA e AES	*EternalBlue*
NotPetya	*Windows*	*file-encrypting*	AES	*EternalBlue*
CryptoWall	*Windows*	*file-encrypting*	RSA	Spam
Ryuk	*Windows*	*file-encrypting*	RSA e AES	Spam e Injeção de código
TeslaCrypt	*Windows*	*file-encrypting*	AES	Angler (Adobe Flash *exploit*)
CTB-Locker	*Windows* e *Linux*	*file-encrypting*	AES e ECC	Spam e Injeção de código
Lilocked	*Linux*	*file-encrypting*	RSA e AES	Vulnerabilidades do sistema

Tabela 2.1: *ransomwares* e suas características

2.3 SOLUÇÕES

Atualmente as soluções utilizadas são simples e preventivas: realizar constantemente a cópia de arquivos importantes e manter essas cópias em outras máquinas ou discos externos que não estejam na mesma rede que a máquina principal. Em termos do cotidiano, realizar backups rotineiros que sejam salvos em outras máquinas, evitando assim que essas cópias possam ser destruídas em conjunto dos arquivos originais pelo ransomware. Esse tipo de solução também deve ser feito junto de ações de cuidado do sistema de maneira geral, como sistemas operacionais atualizados com as atualizações de segurança mais recentes, e evitar a instalação de programas não confiáveis. Essas técnicas, ainda são as principais recomendações (Malwarebytes, 2019) para evitar as perdas que *ransomwares* podem causar.

Apesar de efetivas e simples, essas soluções nem sempre podem ser empregadas com eficácia. Por exemplo, manter os sistemas atualizados nem sempre é possível, visto que existem muitos sistemas em que algum tipo de interrupção de funcionamento pode custar vidas, como é o caso de sistemas de hospitais. Nesse caso, a melhor solução seria o próprio sistema ter algum tipo de ferramenta nativa capaz de realizar a detecção do ataque e impedi-la, como é o caso das propostas a seguir.

Para detecção, alguns antivírus focam na leitura e reconhecimento de assinatura de algumas famílias de *ransomware* para impedir a infecção de computadores. Essa técnica nem sempre é a melhor, visto que a assinatura desses *malware* s muda constantemente, além da existência daqueles que não se encaixam em uma família especifica e passam despercebidos por verificações de sistema.

Em seu mesmo artigo sobre a variedade de *ransomwares* (Kharraz et al., 2015) sugere que uma das possíveis soluções para detectar e impedir um ataque de *ransomware* seria através do monitoramento das requisições de escrita e leitura (I/0) em disco e a proteção da MFT (Tabela de arquivos de sistema) em um sistema de arquivos NTFS.

Na literatura outras propostas também já foram feitas. Por exemplo o software *CryptoDrop* (Scaife et al., 2016), apresenta uma abordagem interessante. Diferente das anteriores que focam somente na detecção do ataque baseado majoritariamente nas requisições de escrita e leitura do disco por um processo, o *CryptoDrop* foca na mudança do estado do arquivo, levando em consideração a mudança de estado de acordo com métricas pré definidas, e assim podendo alertar para um ataque que esteja ocorrendo em tempo real e imediatamente bloqueá-lo. A proposta do *CryptoDrop*, de monitorar mudança em arquivos baseado em métricas e não somente na requisição de escrita e leitura, é uma solução interessante, já que torna possível definir padrões para serem monitorados e verificados, simplificando o processo de monitoramento das chamadas de sistema de escrita.

Similar ao *CryptoDrop*, o Redemption (Kharraz e Kirda, 2017), oferece uma proposta de proteção em tempo real, usando métricas similares às do *CryptoDrop*, como a entropia de Shannon, e a observação de alterações de conteúdo dos arquivos e comportamento de chamadas de sistema. Ele se propõe a identificar ataques em tempo real de *ransomwares* utilizando o que chama de pontos de "malícia" que é feito se baseando em um cálculo com as métricas mencionadas.

Outra solução interessante é o ShieldFS (Continella et al., 2016), um *driver* para *Windows* que cria camadas de *copy-on-write* (COW) para cada processo de escrita no sistema. Através de monitoramento da atividade dos processos, o ShieldFS bloqueia processos suspeitos de atividades maliciosas no sistema, e restaura os arquivos ao estado anterior a essas escritas. O ShieldFS, se baseia no monitoramento do comportamento dos processos, por exemplo, utilizando métricas para analise do número de vezes que um processo executa escritas e leituras.

Apesar de não ser uma solução propriamente dito, o UNVEIL (Kharraz et al., 2016) é uma ferramenta importante de ser listada aqui. O UNVEIL cria um ambiente artificial de usuário monitorado e então observa o comportamento de processos a fim de identificar possíveis *ransomwares* e analisar seu comportamento. A razão pela qual optou-se por adicioná-lo a esta lista é que por mais que ele não seja uma solução direta para o problema de *ransomwares*, o UNVEIL contribui para o desenvolvimento de novas técnicas e métricas para novos *ransomwares* que sejam detectados.

Na tabela 2.2 é apresentada uma relação das soluções existentes mencionadas. Nela podemos observar que não existem muitas opções para sistemas operacionais *Linux*. Esse fato pode ser considerado preocupante em um futuro próximo, lentamente estão ocorrendo mais casos de *ransomwares* específicos para *Linux* (Seals, 2019). Esse fenômeno pode ser atribuído ao fato

que *ransomwares* atualmente têm como principais vitimas organizações privadas (James Wyke, 2015). Em setembro de 2019 foi detectada a campanha de ataques do *ransomware* Lilocked (ou Lilu) (Cimpanu, 2019b) em que o seu principal alvo são justamente servidores *Linux*, mostrando assim que sistemas *Linux* não estão livres do risco de ser atacados por esse tipo de *malware*.

Solução	Sistema Operacional	Momento de Ação	Métricas Utilizadas Para Identificar Ataques	Recupera Arquivos Perdidos?
Cópias de Arquivos (backups)	Todos	Após o Ataque	Perda dos arquivos	Sim
CryptoDrop	*Windows*	Em tempo real (durante o ataque)	Estado dos arquivos	Não
ShieldFS	*Windows*	Em tempo real (durante o ataque)	Comportamento de Processos	Sim
UNVEIL	*Windows*	Em tempo real (durante o ataque)	Comportamento de Processos	Não
Redemption	*Windows*	Em tempo real (durante o ataque)	Comportamento de Processos	Sim

Tabela 2.2: Principais Soluções Para *ransomwares* e Seu Uso

3 CRYPTODROP E LOOKRANSOM

Este capítulo apresenta o desenvolvimento e avaliação uma ferramenta de detecção de ransomware para ambientes Linux chamada *LookRansom*. Inicialmente será apresentado com mais detalhes o *CryptoDrop*, um software feito para sistemas *Windows*, no qual inspirou a criação do *LookRansom*. Em seguida será descrito o *LookRansom*, seu objetivo, detalhes das métricas utilizadas por ele para detecção, seu fluxo de execução e detalhes da implementação em si. Além disso será explicada a implementação do sistema de arquivos Fuse que irá utilizar o *LookRansom* para detecções.

3.1 CRYPTODROP

O *CryptoDrop* é um software para sistemas *Windows* em duas versões, uma gratuita e outra empresarial (paga). A versão inicial do *CryptoDrop* foi apresentada em 2016 no artigo ***CryptoLock (and Drop It):Stopping Ransomware Attacks on User Data*** (Scaife et al., 2016). A proposta do software é detectar ataques ransomware em tempo real. Após ser feita uma chamada de sistema de alteração de algum arquivo (escrita, leitura, apagar o arquivo), o *CryptoDrop* é notificado e analisa a operação, monitorando alterações do estado de arquivo através de cinco métricas: Entropia de Shannon; Fuzzy Hash do conteúdo do arquivo, utilizando o algoritmo sdhash; Tipo do arquivo; Comportamento de Processos durante manipulação de arquivos; Arquivos sendo apagados e recriados em grande número. As métricas foram definidas baseadas no comportamento comum de ransomwares, e tiveram resultados interessantes, sendo alguns desses resultados a detecção de ransomwares em si com baixo número de falsos positivos (Scaife et al., 2016).

Apesar do *CryptoDrop* considerar como principais métricas o estado do arquivo (Entropia, Hash de conteúdo e tipo do arquivo), as métricas secundárias como comportamento de processos quando eles estão alterando esses estados também são importantes. Esse monitoramento permite que o *CryptoDrop* identifique ransomwares que não somente escrevem arquivos, mas também aqueles que apresentam apenas um comportamento destrutivo como a exclusão dos arquivos originais e substituição dele por arquivos cifrados que contenham o mesmo nome (Scaife et al., 2016). Essas métricas adicionais garantem que a detecção do *CryptoDrop* não irá gerar falsos positivos e que também irá conseguir detectar mais de uma familia de ransomware (Scaife et al., 2016).

A figura 3.1, obtida do artigo (Scaife et al., 2016), mostra a arquitetura do *CryptoDrop*, exemplificando de maneira visual seu funcionamento. Ele funciona como um *driver*, filtrando as chamadas e verificando arquivos. Uma das razões para o *CryptoDrop* funcionar dessa maneira, se deve ao fato do comportamento de ransomwares em si, pois diferente de outros softwares maliciosos, só é possível constatar que o ataque ocorreu após pelo menos um dos arquivos do usuário terem sido cifrados. A estratégia do *CryptoDrop* de verificar um arquivo assim que ele tenha sido alterado garante que, caso seja identificado imediatamente o ataque, o mesmo será interrompido, garantindo a segurança do arquivos restantes no sistemas. O que fornece dessa maneira uma solução em tempo real para o problema de ataque de ransomwares.

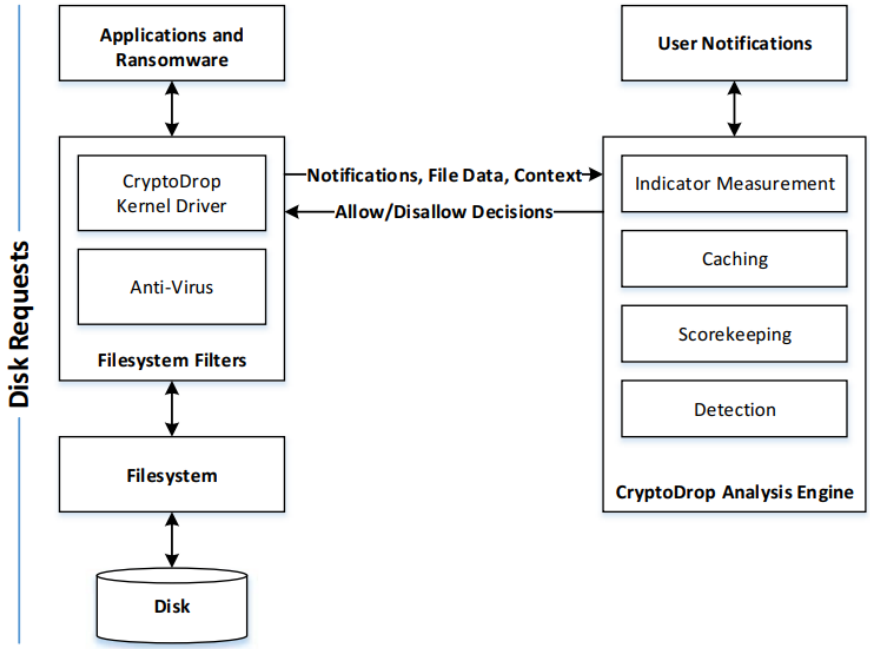


Figura 3.1: Arquitetura do *CryptoDrop* (Scaife et al., 2016)

3.2 LOOKRANSOM

O *LookRansom* é um programa capaz de verificar se um arquivo foi cifrado. Semelhante ao *CryptoDrop*, o *LookRansom* verifica arquivos após a execução de chamadas de escrita. A ideia é que o programa seja implementado não como um *driver*, mas como uma função extra na chamada de sistema, podendo assim auxiliar um sistema de arquivos a verificar a integridade de um arquivo antes do usuário perceber problemas. A verificação de estado do arquivo é feita utilizando informações com relação ao estado do arquivo, sendo as métricas salvas em arquivos auxiliares.

As informações de estado utilizadas para a detecção são: Entropia de Shannon do arquivo; Tipagem (nesse caso *magic number*, e por fim o hash de seu conteúdo (*Fuzzy Hash*). A razão do uso dessas métricas é que as mesmas são utilizadas pelo *CryptoDrop* para suas verificações e se mostraram efetivas (Scaife et al., 2016). Apesar de utilizarem outras métricas, além destas, neste trabalho optou-se por usar somente estas métricas, porque o *LookRansom* deve ser executado a cada chamada de sistema, inviabilizando o uso de algoritmos mais pesados.

A proposta da utilização do *LookRansom* é integrada ao sistema de arquivos, como uma ferramenta nativa do sistema, e não como um programa independente. A razão disso é que seria mais interessante ter um sistema que por si só detecta e consegue parar ataques ransomwares, criando assim um sistema de arquivos "inteligente". A escolha de implementar essa funcionalidade no sistema Linux em vez do *Windows* é consequência do fato de ransomwares ainda não serem um problema resolvido em sistemas Linux, dando possibilidade de experimentos na área. Por exemplo, para *Windows* já existem propostas semelhantes ao *LookRansom*, sendo uma dela o próprio *CryptoDrop* e a outra o ShielfFS, que tem uma ideia semelhante a desse trabalho.

Como mencionando anteriormente, apesar do desejo de uma implementação do *LookRansom* com um sistema de arquivos padrão do Linux (Btrfs por exemplo), optou-se pelo uso do Fuse por questões de praticidade e simplificação da implementação e testes (Wikipedia, 2017)(libfuse, 2001).

Para ter uma detecção rápida no ataque de um ransomware, é necessário que a detecção ocorra assim que o primeiro arquivo do usuário for cifrado, conforme foi visto no artigo (Scaife et al., 2016). Boa parte dos ransomwares quando iniciam seu ataque, fecham processos que o usuário possa utilizar para interromper o ataque (Kharraz et al., 2015). Essa é uma técnica de evasão clássica nesse tipo de malware. Conseguir fazer um detector que não possa ter seu processo interrompido pelo ransomware se torna importante.

A ideia de ter um detector na execução, integrado ao sistema de arquivos, vem justamente em consequência dessa tentativa de evasão. Se o mecanismo de detecção é algo nativo da chamada de sistema, ainda mais sendo essa chamada de sistema necessária para o ransomware executar a cifragem de arquivos. Torna-se impossível para o ransomware evadir-se dessa técnica de detecção.

Outro fato positivo no uso dessa solução é que ela dispensa verificações extensas de todos os arquivos do sistema atrás de padrões. Essa verificação é um comportamento comum em anti-vírus e que se mostrou ineficaz na identificação de ransomwares por parte dessas ferramentas.

A verificação do estado do arquivo no exato momento em que é encerrada a sua escrita elimina essa necessidade de verificações em massa, pois assim que o primeiro arquivo que fosse cifrado for fechado, o *LookRansom* imediatamente avisaria o problema e bloquearia o processo suspeito de continuar suas escritas.

Para conseguir fazer a inclusão desse detector nas chamadas de escrita de um sistema de arquivo, sem precisar implementar o código a nível do *kernel*, optou-se pelo uso do Fuse como sistema de arquivos. Foi feita essa escolha devido à simplicidade de implementação de um sistema de arquivos em Fuse e à facilidade de incluir o *LookRansom* em sua chamada de escrita, sem necessariamente ter que recompilar o *kernel* e executá-lo para realizar testes. Por esse mesmo motivo de praticidade optou-se por utilizar a linguagem Python tanto na implementação do *LookRansom* quanto Fuse, por preferências pessoais. Como foco era a construção de um protótipo de detector e não necessariamente um detector com máxima eficiência, não foram encontrados problemas no uso de uma linguagem interpretada.

3.2.1 Fuse - Filesystem in Userspace

O Fuse, *filesystem in userspace* (Wikipedia, 2017), é uma biblioteca de interface para sistemas Unix que torna possível usuários não privilegiados criarem seus próprios sistemas de arquivos, realizando tudo isso em nível de usuário. O Fuse é um *software opensource*, segue a licença GNU e está presente para uso em qualquer distribuição Linux.

Para um usuário de sistema criar um sistema Fuse, ele deve escrever um código utilizando a biblioteca libFuse (libfuse, 2001), que está disponível para uso em várias linguagens de programação, como C e Python. A biblioteca oferece uma API com as principais funções de um sistema de arquivos, o programador deve então definir essas operações, e também definir um ponto de montagem para o seu sistema Fuse.

Montado esse sistema definido no código, o modulo Fuse irá intermediar a comunicação entre o *kernel* e o sistema de arquivos escrito pelo programador. Os comandos executados no sistema Fuse serão redirecionados pelo *kernel* do sistema operacional para o módulo Fuse através do VFS (*virtual file system*) e então o Fuse irá executar esse comando conforme ele foi implementado pelo programador. Por exemplo se o programador não definiu a implementação

do comando *ls* em seu sistema Fuse, o comando *ls* não irá funcionar em seu sistema de arquivos. A figura 3.2 mostra de maneira visual esse fluxo.

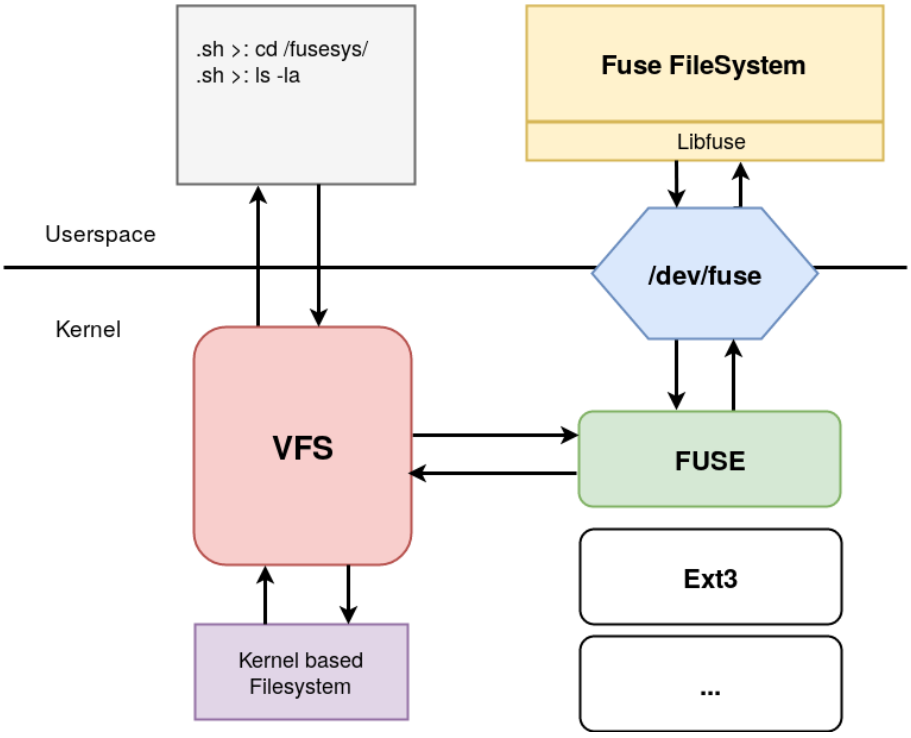


Figura 3.2: Modelo de funcionamento do Fuse (Wikipedia, 2017)

Alguns dos sistemas de arquivos virtuais que utilizam o Fuse como base para sua implementação são: EncFS, um sistema de arquivos capaz de utilizar diretórios criptografados; GlusterFS, um sistema de arquivos clusterizável feito para dados em grande escala e o MooseFS, um sistema de arquivos à prova de falhas e alto desempenho.

No caso da implementação deste trabalho, a biblioteca fuse usada é o Fusepy, ou seja, a biblioteca Fuse para a linguagem Python. O sistema implementado é somente um modelo simples que define a maior parte de seus comandos como os comandos padrão do sistema, sendo o único método alterado para a implementação, o método de *release* dos arquivos, justamente por esse ser, de acordo com o manual da API do Fuse, o último método chamado na escrita de um arquivo e que executa o fechamento do link descritor do arquivo.

No final da função de *release* é verificado se o arquivo que foi fechado existe (por exemplo, não é um arquivo temporário) e se esse arquivo não é um diretório. Caso essas condições sejam verdadeiras, é verificado se esse arquivo aberto é um arquivo novo criado. Se sim, então é criado o arquivo de métricas no formato `.mm` que é salvo em um diretório também criado na execução do Fuse, chamado `files_info`. Esse diretório está localizado fora do sistema de arquivos de usuário e em uma pasta criada no momento de execução do Fuse, a razão disso é uma tentativa de impedir que os próprios arquivos de métricas também sejam cifrados pelo ransomware, podendo assim causar falhas na detecção.

A figura 3.3 representa visualmente o fluxo e modelo completo do detector sendo executado durante as chamadas de sistema de escrita do Fuse. O usuário ou um processo chama o processo de escrita de um arquivo, e caso esse arquivo seja um arquivo comum (imagem, texto etc), ao concluir o processo de escrita o método de release é ativado.

Nesse momento, o Fuse chama a função de detecção (LookRansom) para verificar a estado atual desse arquivo, comparando com o estado anterior que ele se encontrava antes da

chamada de escrita. Caso a verificação de pelo menos duas das três métricas utilizadas indique que existe algum problema, o *LookRansom* irá bloquear o processo que realizou essa alteração, tirar as permissões de seu executável (caso ele exista) e move-lo para um diretório de quarentena em que os arquivos não podem ser executados.

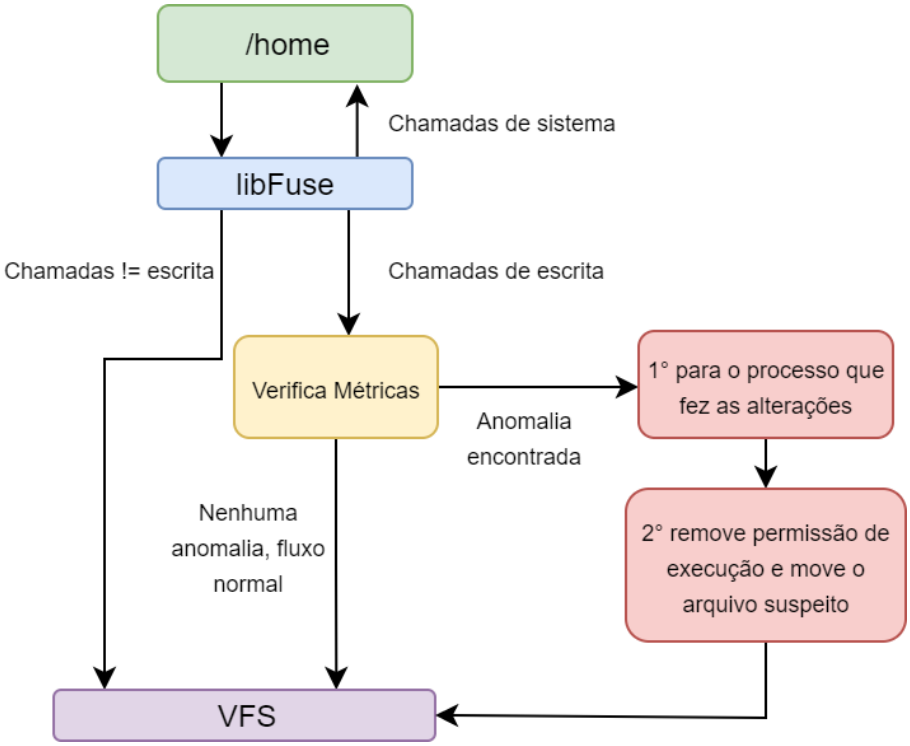


Figura 3.3: Modelo visual do funcionamento do *LookRansom* junto do Fuse

Caso esse arquivo seja um arquivo novo, o *LookRansom* simplesmente cria o arquivo de métricas desse novo arquivo e o salva para futuramente realizar as comparações.

3.2.2 Métricas

As métricas escolhidas para realizar a comparação de estado dos arquivos, foram baseadas nas métricas utilizadas no próprio *CryptoDrop*: Entropia de Shannon, hash de conteúdo (*Fuzzy Hash*) e por fim tipagem do arquivo (*magic numbers*). Nos resultados apresentados pelo artigo (Scaife et al., 2016), somente essas métricas já conseguem bons indicativos de detecção. Como a ideia do *LookRansom* é detectar utilizando o mínimo de recursos possíveis, optou-se por utilizar apenas esses três métodos para verificação.

3.2.3 Entropia de Shannon

De maneira simplista podemos definir entropia como a medida de aleatoriedade. No caso da entropia de Shannon, em termos técnicos, podemos defini-la como a medida de incerteza de uma probabilidade de distribuição de tamanho M (Kim, 2018).

Considerando $P(i)$ a probabilidade de um evento i ocorrer, a entropia de Shannon $H(x)$ é definida por:

$$H(X) = -\sum_{i=1}^{M} P(i)\, log_2 \frac{1}{P(i)} \tag{3.1}$$

Uma maneira de interpretar a entropia de Shannon é que arquivos cifrados e comprimidos (arquivos no formato `tar` ou `zip` por exemplo) tendem a ter uma entropia alta, enquanto arquivos comuns (imagem, texto), tendem a ter uma entropia baixa. Esse fator é extremamente útil quando utilizado para verificação de ataques ransomware em ferramentas como o Redemption (Kharraz e Kirda, 2017) e *CryptoDrop* (Scaife et al., 2016), já que a mudança de entropia em um arquivo de baixa para alta, é um indicador forte de que o arquivo provavelmente foi cifrado. Por essa razão o *LookRansom* também usa a entropia de Shannon para verificar o estado dos arquivos.

3.2.4 Hash de Conteúdo com ssdeep

Uma função de *hash* tem como entrada uma sequência de bytes de tamanho variável e gera como saída uma sequência de bytes de tamanho fixo única para cada entrada. Como o *hash* gerado pelo conteúdo de um arquivo é único, ele pode ser utilizado como uma assinatura desse arquivo.

Justamente por cada arquivo gerar um *hash* diferente conforme seu conteúdo, caso um arquivo tenha seu conteúdo alterado, o *hash* deve mudar completamente. Para conseguirmos realizar comparações entre arquivos e verificar a semelhança se torna necessário o uso de algoritmos de *hash* capazes de medir a diferença entre arquivos, nesse caso o algoritmo escolhido foi o *ssdeep* (Kassner, 2011). O *ssdeep* é conhecido como um algoritmo do tipo *Fuzzy Hash*, ele é capaz de partir da *hash* gerada de dois arquivos, estimar a similaridade entre eles. Esse tipo de algoritmo é principalmente utilizado para identificação e analise de *malware* (Sarantinos et al., 2016) (Kassner, 2011). Uma curiosidade é que o próprio VirusTotal utiliza o *ssdeep* e *sdhash* (variações de algoritmos de *Fuzzy Hash*) como método de analise de semelhança nos malwares listados por ele .

Por já existir uma biblioteca com uma implementação funcional e efetiva do *ssdeep* para Python, optou-se por utilizá-la para a criação e comparação de *hash* de conteúdo de arquivos. Além disso o fato desse algoritmo ser usado por ferramentas como o VirusTotal, prova a funcionalidade e eficácia dele, o tornando uma escolha segura para uso no *LookRansom*.

3.2.5 Tipo do arquivo

A identificação da tipagem de um arquivo não é definida somente pela sua extensão (`.txt`, `.png`, etc) que fica localizada no final de seu nome, na realidade a fonte mais confiável para indicar o tipo de um arquivo é o *magic number*. O *magic number* de um arquivo, consiste dos três primeiros bytes de um arquivo, que são únicos para cada tipo e indicam a assinatura do arquivo (Scaife et al., 2016).

Arquivos que são cifrados perdem a informação com relação ao seu *magic number*, já que seus bytes iniciais são alterados junto do resto do arquivo. Por esse motivo o *LookRansom* verifica se houveram alterações no tipo do arquivo verificando o seu *magic number*. Por ser um indicador claro de problemas (já que dificilmente uma imagem irá deixar de ter o *magic number* de imagem para ter o *magic number* de um arquivo de texto por exemplo), foi optado por utilizar essa métrica para comparações.

3.2.6 Problemas Conhecidos

Apesar de interessante, alguns problemas já podem ser notados nessa implementação. Mesmo que o algoritmo de detecção seja otimizado para ser executado rapidamente, as chamadas de escrita terão um pequeno *overhead* no fim de sua execução, considerado que serão feitas

verificações toda vez que essas chamadas forem concluídas. O que reduziria o desempenho em um sistema que realize muitas escritas em arquivos.

Outro problema é no aumento do número de arquivos nesse sistema, já que todas as métricas são escritas em arquivos isolados em outros diretórios do sistema, em um sistema de arquivos com N arquivos, teríamos 2N arquivos, pois para cada um desses arquivos, teria outro de mesmo nome com suas métricas. Além disso, a maior falha possível para essa detecção se encontra justamente na necessidade de salvar as informações de estado de cada arquivo, pois caso esses arquivos também sejam cifrados, o detector não irá funcionar como devido. Estes são os principais pontos negativos dessa implementação para detecção.

4 EXPERIMENTOS E RESULTADOS

Esse capitulo tem por foco explicar como foram realizados os experimentos e resultados obtidos e também acertos e erros. Por questões de segurança e praticidade, todos os testes foram executados em uma máquina virtual utilizando o VirtualBox. A máquina tem como sistema operacional instalado o Ubuntu 18.04 LTS. Seu sistema de arquivos raiz é o Btrfs.

As configurações de hardware da máquina virtual são um processador virtual dual core e memória RAM de 4 GB. O planejamento era que caso o sistema tivesse um bom desempenho nos testes unitários e básicos, seriam então realizados os testes com ransomwares reais para Linux como o Lilocker. Conforme será explicado mais adiante, o sistema Fuse não funcionou em conjunto do detector. Então o planejamento foi refeito e foram abandonados os planos de teste envolvendo ransomwares reais, focando somente os testes unitários do *LookRansom* em si sem estar inserido no sistema de arquivos Fuse.

4.1 TESTES FUSE E LOOKRANSOM

Os primeiros testes realizados, foram justamente o modelo completo do sistema de arquivos com o *LookRansom*, após verificar que ambas as implementações estavam prontas (o *LookRansom* criando e comparando os estados dos arquivos e o Fuse executando como um sistema de arquivos normal). O código do *LookRansom* foi inserido no método de *release* do Fuse, já que a última função a ser chamada durante a escrita do arquivo no Fuse é justamente a função de *release* que fecha o descritor do arquivo aberto e permite que ele possa ser utilizado por outros processos. Entretanto essa função era executada de maneira assíncrona no Fuse, muitas vezes sendo chamada antes do arquivo em que estava sendo escrito ser fechado. Dessa forma o *LookRansom* era executado antes do momento correto, o que causava inconsistências no sistema de arquivos. Optou-se então por iniciar um processo de *debug* no sistema de arquivos Fuse antes de fazer outras escolhas ou optar por testes diferentes.

4.1.1 Fusepy e Problemas

Para entender melhor os problemas que ocorreram é necessário explicar com mais detalhes como foi feita a implementação do Fuse. Ao utilizar a API oferecida pelo Fusepy, pode-se optar em implementar algumas chamadas de sistema do zero ou implementar elas utilizando de base chamadas padrões do *kernel*. Por exemplo, para a escrita o programador pode implementar a função *write* do Fuse como um método que chama a função *write* do sistema operacional. Na implementação feita para o Fuse com o *LookRansom*, são utilizadas ambas as estratégias mencionadas. Funções que não precisam ser alteradas, chamam as funções nativas do sistema como `os.open`, enquanto outras funções como a de escrita, foram alteradas conforme necessário. A figura 4.1 mostra de maneira visual como são feitas as chamadas de função da API do fuse conforme a operação do sistema de arquivos.

Como é visto na figura, todas as operações que envolvem leitura e escrita de arquivos, chamam no fim do processo a função *release*. Essa função fecha o descritor do arquivo permitindo que ele possa ser usado por outros processos. No modelo implementado o detector *LookRansom* foi implementado dentro da função de *release* logo após ela chamar a função `os.close` que fecha o descritor de arquivo. Dessa maneira seria garantido que o *LookRansom* somente abriria o arquivo que foi escrito, após o processo de escrita ter sido encerrado. Mas isso não foi o

ocorrido. O sistema Fuse chamava a função *release* antes do arquivo fechar. O que resultava em um travamento do sistema de arquivos que parava de responder comandos e um travamento no processo de escrita em si que parava de funcionar, porém não era interrompido, criando um *deadlock* nesse sistema. A causa do *deadlock* é justamente a biblioteca do Fusepy por motivos desconhecidos executar no momento indevido a função de release, chamando ela, por exemplo logo após a chamada de abertura do arquivo e não no fim da chamada. Outro problema é que apesar da execução da função de encerramento do descritor do arquivo, o mesmo continuava aberto.

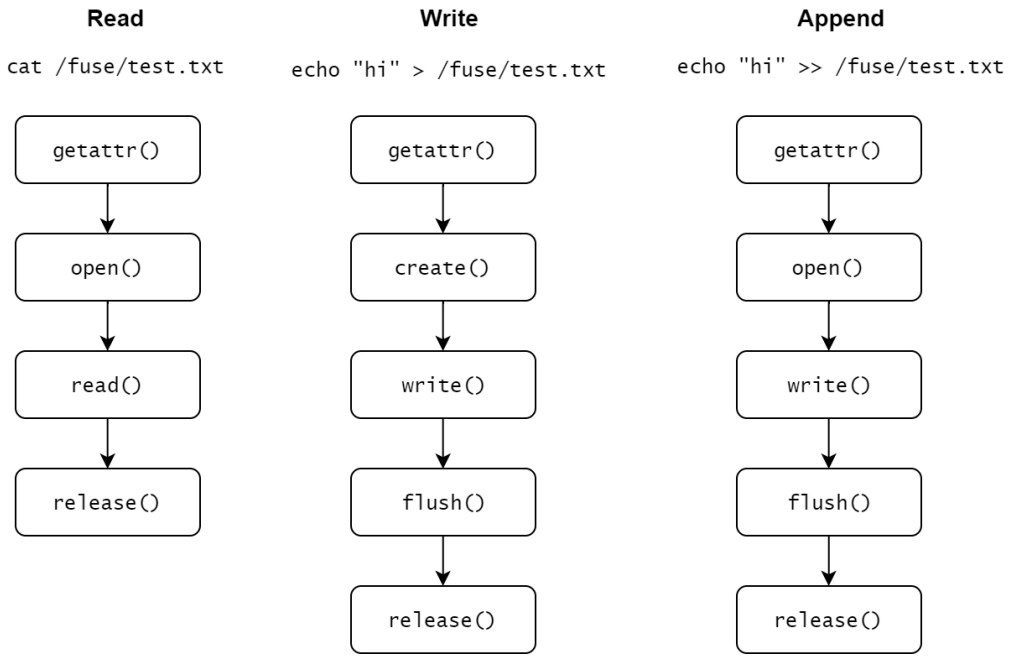


Figura 4.1: Ordem de Execução das Funções Fuse (Wikipedia, 2017)

E então foi iniciado um processo de modificar ainda mais as chamadas e tentar utilizar menos chamadas de nativas de sistema como `os.close` para a implementação de escrita e leitura. Porém, os problemas continuaram, avisos de que o sistema de arquivos estava sem espaço para escrita, a não criação de arquivos de estado mesmo com a alteração ou criação de um novo arquivo, a não ativação do *LookRansom* mesmo após a criação e alteração de um arquivo, e por fim *deadlock* como visto anteriormente. Esses são alguns dos problemas que surgiram durante essa fase de modificações e inserção do *LookRansom* em outros momentos da chamada de escrita. Durante esse período foi possível observar limitações na API do Fusepy com relação à modificações mais complexas quando se trata de fluxo de operações. Além da descoberta de *bugs*, como a execução assíncrona das funções mesmo com esta opção desativada, função de contexto perdendo informações durante troca de funções e dessa maneira não sabendo mais o PID do processo que chamou a operação e assim tornando um objeto nulo, sistema levantando mensagens de erro e aviso quando não existe o erro referente a mensagem, entre outros.

Por fim após pesquisar sobre esses problemas junto à comunidade de desenvolvedores usuários da biblioteca Fusepy, descobriu-se que esta tem vários problemas de implementação e está sendo descontinuada (GitHub, 2019). E devido ao tempo curto para alterações após o longo período gasto em testes, desenvolvimento do Fuse e *LookRansom* e *debug* de código. Optou-se por abandonar os experimentos utilizando o sistema de arquivos e focar somente nos testes de comparações do *LookRansom* em si, visto que não teria tempo hábil para realizar a portabilidade de toda implementação feita até o momento de linguagem Python para a linguagem C.

4.2 TESTES UNITÁRIOS

Sem poder utilizar o *LookRansom* em um sistema de arquivos em si para testar na prática sua capacidade de detecção, sobrou somente a escolha de cifrar arquivos de diferentes formatos via *script* e verificar suas comparações. Então foi reunido um total de 60 arquivos de três diferentes formatos para serem criadas as métricas e testados, desses 60 arquivos, 20 no formato `txt` com o seu conteúdo sendo números gerados de maneira aleatória (função `rand()`), 20 imagens no formato `jpg`, 20 arquivos no formato `pdf`, livros escolhidos de maneira aleatória do Project Gutenberg. Os arquivos com um tamanho acima de 10MB em média eram os arquivos `pdf`, porém nenhum arquivo testado é maior que 30MB. O *LookRansom* foi executado de maneira à criar os arquivos de estado de cada um dos arquivos de teste, sendo o arquivo de estado constituído do nome do arquivo ao qual se refere e o formato `.mm` a fim de padronização, com os estados referente a cada um desses arquivos. Os estados são somente 3 linhas de informação, sendo a primeira a entropia de Shannon calculada do conteúdo do arquivo, a segunda o *fuzzy hash* também calculado a partir do conteúdo do arquivo e por fim na última linha *magic number* desse arquivo.

Por questões de conveniência e da maneira que foi implementado o *LookRansom*, foram definidos 3 tipos de falsos positivos diferentes, o falso positivo no qual somente uma das métricas teve uma alteração suspeita no estado do arquivo, o no qual duas e por fim o falso positivo de maior preocupação, no qual todos os estados de comparação falharam.

A comparação dos estados dos arquivos e testes em si teve duas fases, a primeira se tratou de alterações normais no conteúdo, como por exemplo, inserir mais bits nos arquivos de imagem, texto e `pdf` e verificar se uma das métricas sofreria alterações, enquanto a segunda foi testes com os arquivos após serem cifrados.

4.2.1 1 - Fase: Alterações Comuns

Um ponto importante na detecção é ter o menor número de falsos positivos possível, para isso é preciso garantir que apesar de uma escrita normal, o detector não irá encontrar problemas que na verdade só são arquivos alterados por um usuário qualquer. Durante essa fase de testes foi inserido em cada arquivo um vetor de strings de tamanho 100, contendo em todos seus índices a letra ***A***. Para evitar a perda do *magic number* localizado no inicio do arquivo, causando um aviso de falso positivo de nível 1, essa *string* foi somente inserida no final do arquivo. Com esse teste absolutamente nenhum falso positivo ocorreu.

Para sobrecarregar um pouco foi então aumentado o tamanho dessa string inserida no final para 2000 caracteres. Apesar do *hash* de conteúdo a esse ponto apresentar pequenas alterações, o indicador ainda apontava que o arquivo anterior e o atual mantinham um nível de semelhança acima de 21%. De acordo com o manual do *ssdeep*, arquivos que apresentem uma pontuação de 21% a 100% na comparação são arquivos semelhantes. Por esse motivo, essas comparações foram baseadas com um limite de até 21% para a diferença do conteúdo. Apesar desses testes serem simples, já demonstram que o algoritmo não tem falhas com alterações simples e pequenas, apontando um bom nível de confiabilidade nas comparações.

4.2.2 2 - Fase: Comparação Arquivos Cifrados

Para cifrar os arquivos originais, foi utilizada uma implementação do algoritmo de cifragem RSA-2048, visto que ele é comumente usado em ransomwares. A implementação foi escrita em linguagem Python, o algoritmo cifrou todos os arquivos de texto, imagem e `pdf`

que tinham as métricas anteriores. Após a cifragem desses arquivos, foi aplicado o método de comparação de métricas do *LookRansom*.

Em absolutamente todos os casos ele alertou que ocorreram pelo menos duas alterações nos estados dos arquivos que por fim indicasse que o arquivo foi cifrado. Para arquivos de texto, somente dois indicadores do *LookRansom* foram úteis, o de hash e entropia de Shannon. Porém para com relação a *magic number* ele ainda alertou que era um arquivo de texto, o que faz sentido visto que a cifragem gera um arquivo que pode ser interpretado como arquivo de texto.

Para arquivos no formato `pdf` em poucos casos (3 arquivos) o indicador de entropia não foi alertado, ou seja, a entropia do arquivo em si já era extremamente elevada a ponto de nas comparações o *LookRansom* não conseguir diferenciar; Com relação a arquivos de imagem (`jpg`) todas as métricas alertaram alterações.

A Tabela 4.1 apresenta os resultados dos testes com o *LookRansom* comparando estados de arquivos após eles terem sido efetivamente cifrados.

Tipo de Arquivo	Qtde Total de Arquivos	Alteração de Entropia	Alteração de Tipo	Alteração Hash	Alertas de Alteração
Texto (txt)	20	20	0	20	20
Imagem (jpg)	20	20	20	20	20
PDF	20	17	20	20	20

Tabela 4.1: Resultados de Comparação *LookRansom*

4.3 RESULTADOS FINAIS

Apesar das dificuldades encontradas no uso do Fuse e no fato de sua implementação não ter sido efetivamente útil para os testes finais, os resultados foram relativamente positivos. Nos testes unitários do algoritmo, o *LookRansom* não apresentou falsos positivos, embora seja necessário frisar que os testes focados para isso foram extremamente simples e talvez sejam necessários testes mais apurados para verificar melhor a questão de falsos positivos.

Com relação à detecção de anomalias na segunda análise, que é a comparação com arquivos cifrados em si, ele detectou esse problema em todos os cenários, apesar de nem sempre indicar isso com todas as métricas e sim somente com duas.

Mesmo com o resultado decepcionante dos testes envolvendo o sistema de arquivos Fuse, o *LookRansom* e sua verificação de estados se mostraram efetivos podendo ser implementado não necessariamente somente nesse cenário. Contudo para realmente afirmar isso seriam necessários mais testes em outros cenários e efetivamente um teste real com ransomwares, o que nas condições atuais não é possível.

5 CONCLUSÃO

Este trabalho apresentou a proposta e construção de um sistema de detecção e contenção de ransomwares chamado *LookRansom*. O sistema foi parcialmente construído e testado, com resultados promissores. Apesar de não ter sido possível a execução de um teste real com ransomwares no modelo proposto de detecção, os resultados foram relativamente positivos.

Foi possível verificar que as métricas escolhidas e apresentadas realmente forneciam uma boa base para comparações para identificar se um arquivo foi cifrado ou somente alterado, resultando uma porcentagem muito pequena de falsos positivos. Sendo consequentemente a maior parte de falso positivos, resultantes deles em arquivos com uma entropia alta, podendo assim melhorar a análise dessa métrica em especifico. Também foi possível verificar que apesar de as métricas a serem utilizadas para comparação serem simples e de fácil manuseio, a implementação efetiva de um detector em um sistema de arquivos em si não é trivial, sendo algo que necessita de pesquisa e estudo mais cuidadoso.

Também é possível concluir que para a implementação do *LookRansom* de maneira funcional, ele obrigatoriamente não pode ser em uma linguagem interpretada e de alto nível, como foi proposto neste trabalho, e sim em uma linguagem mais apropriada ao *kernel*, visto que para uma implementação funcional seria necessário alterações a nível de *kernel* no sistema de arquivos. Além disso, é necessário melhorar as questões de otimização desse algoritmo, já que boa parte dos testes foram feitos em escritas de arquivos de tamanhos relativamente pequenos, e não se tem uma base do quão custoso seriam essas comparações e calculo de entropia em arquivos de tamanho superior a 1GB; Isso sem mencionar as alterações que o *LookRansom* teria que sofrer para não encontrar os mesmos problemas que foram encontrados durante seu uso em conjunto do sistema de arquivos Fuse.

Mesmo com as dificuldades encontradas durante o desenvolvimento deste trabalho, a ideal final é que, apesar das dificuldades, existem maneiras de detectar ransomwares e elas podem ser simples, mesmo que o modelo aqui proposto não tenha funcionado como o desejado. É possível observar que as causas de não funcionamento do *LookRansom* são devido a escolhas mal feitas e implementações pouco cuidadosas modelo proposto, e não por culpa da grande capacidade de evasão dos ransomwares.

Alguns dos resultados mostrados ainda são interessantes e poderiam ser aplicados de alguma maneira para detecção em programas ou sistemas operacionais em si, sendo assim o *LookRansom* um modelo conceitual em que muito pode ser melhorado, aplicado de alguma maneira na pratica.

Apesar de não ser mencionado nesse trabalho por não entrar no escopo proposto. É importante mencionar a restauração dos arquivos perdidos. Em um modelo concreto de implementação do *LookRansom* seria interessante a execução dessa verificação em um sistema de arquivos com capacidade de “auto cura”, como o Btrfs.

O Btrfs é um sistema de arquivos que segue o modelo de *copy-on-write*, criando camadas do sistema, sendo sempre possível restaurar arquivos que tenham sido corrompidos ou destruídos de alguma maneira. Em uma aplicação futura seria interessante um detector trabalhar em conjunto com um sistema de arquivos capaz de restaurar arquivos que foram alterados por ransomwares. Essa seria sem dúvidas uma solução interessante e nativa, porém fica apenas como uma reflexão de possíveis implementações e casos em que o *LookRansom* poderia ser utilizado.

REFERÊNCIAS

Abrams, L. (2019). Lilocked ransomware actively targeting servers and web sites. https://www.bleepingcomputer.com/news/security/lilocked-ransomware-actively-targeting-servers-and-web-sites/. Acessado em 08/12/2019.

Al-rimy, B., Maarof, M. e Shaid, S. M. (2018). Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security Elsevvier*, 74:144–166.

Andrew Brandt, B. C. (2019). Sophoslabs 2019 thread report. Relatório técnico, SophosLabs, Boston. USA.

Baker, M. (2017). How cryptocurrencies are fueling ransomware attacks and other cybercrimes. https://www.forbes.com/sites/forbestechcouncil/2017/08/03/how-cryptocurrencies-are-fueling-ransomware-attacks-and-other-cybercrimes/#46a3d1e93c15. Acessado em 23/10/2019.

Bertozzi, M. (2011). Python fuse. https://www.slideshare.net/matteobertozzi/python-fuse. Acessado em 30/11/2019.

Cimpanu, C. (2019a). Over 500 us schools were hit by ransomware in 2019. https://www.zdnet.com/article/over-500-us-schools-were-hit-by-ransomware-in-2019/. Acessado em 30/11/2019.

Cimpanu, C. (2019b). Thousands of servers infected with new lilocked (lilu) ransomware. https://www.zdnet.com/article/thousands-of-servers-infected-with-new-lilocked-lilu-ransomware/. Acessado em 30/11/2019.

Continella, A., Guagnelli, A., Zingaro, G., Pasquale, G., Barenghi, A., Zanero, S. e Maggi, F. (2016). Shieldfs: a self-healing, ransomware-aware filesystem. páginas 336–347.

GitHub (2019). Project status. https://github.com/fusepy/fusepy/issues/134. Acessado em 03/12/2019.

Greenberg, A. (2018). How cryptocurrencies are fueling ransomware attacks and other cybercrimes. https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/. Acessado em 23/10/2019.

Hanel, A. (2019). Big game hunting with ryuk: Another lucrative targeted ransomware. https://www.crowdstrike.com/blog/big-game-hunting-with-ryuk-another-lucrative-targeted-ransomware/. Acessado em 30/11/2019.

James Wyke, A. A. (2015). The current state of ransomware. Relatório técnico, SophosLabs, Boston. USA.

Kassner, M. (2011). Fuzzy hashing helps researchers spot morphing malware. https://www.techrepublic.com/blog/it-security/fuzzy-hashing-helps-researchers-spot-morphing-malware/. Acessado em 03/12/2019.

Kharraz, A., Arshad, S., Mulliner, C., Robertson, W. e Kirda, E. (2016). Unveil: A large-scale, automated approach to detecting ransomware. Em *Proceedings of the 25th USENIX Security Symposium*, páginas 757–772, Austin - TX.

Kharraz, A. e Kirda, E. (2017). Redemption: Real-time protection against ransomware at end-hosts. páginas 98–119.

Kharraz, A., Robertson, W., Balzarotti, D. e Kirda (2015). Cutting the gordian knot: A look under the hood of ransomware attacks. Em *Detection of Intrusions and Malware, and Vulnerability Assessment*, páginas 2–24, Milan - Italy.

Kim, A. (2018). The intuition behind shannon's entropy. `https://towardsdatascience.com/the-intuition-behind-shannons-entropy-e74820fe9800`. Acessado em 03/12/2019.

libfuse (2001). Libfuse. `https://github.com/libfuse/libfuse`. Acessado em 03/12/2019.

Lima, R. (2019). Ransomware ryuk força prosegur a desativar departamento de ti. `https://www.tecmundo.com.br/seguranca/148215-ransomware-ryuk-forca-prosegur-desativar-departamento-ti.htm`. Acessado em 30/11/2019.

Malwarebytes (2019). All about ransomware. `https://www.malwarebytes.com/ransomware/`. Acessado em 02/12/2019.

MalwarebytesLabs (2019). 2019 stage of malware. Relatório Técnico ISBN 9788584800025, MalwarebytesLabs, Madrid. Spain.

Palmer, D. (2017). Wannacry ransomware: Hospitals were warned to patch system to protect against cyber-attack - but didn't. `https://www.zdnet.com/article/wannacry-ransomware-hospitals-were-warned-to-patch-system-to-protect-against-cyber-attack-but-didnt/`. Acessado em 23/10/2019.

Petters, J. (2018). The state of cryptowall in 2018. `https://www.varonis.com/blog/cryptowall/`. Acessado em 30/11/2019.

Reis, M. (2017). Wannacry: o primeiro ransomworm na indústria de cibersegurança. `https://www.proof.com.br/blog/wannacry-ransomworm/`. Acessado em 23/10/2019.

Sarantinos, N., Benzaid, C., Arabiat, O. e Al-Nemrat, A. (2016). Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities. páginas 1782–1787.

Scaife, N., Carter, H., Traynor, P. e Butler, K. R. (2016). Cryptolock (and drop it): Stopping ransomware attacks on user data. Em *2016 IEEE 36th International Conference on Distributed Computing Systems*, páginas 303–312, Nara - Japan.

Scammell, R. (2019). Wannacry is dying but eternalblue is alive and kicking a million machines. `https://www.verdict.co.uk/wannacry-eternalblue-rise/`. Acessado em 30/11/2019.

Seals, T. (2019). Unusual linux ransomware targets nas servers. `https://threatpost.com/linux-ransomware-nas-servers/146441/`. Acessado em 03/12/2019.

Security, P. (2017). Wannacry report. Relatório técnico, Panda Security, Santa Clara. CA.

Wikipedia (2008). Aids (trojan horse). https://en.wikipedia.org/wiki/AIDS_(Trojan_horse). Acessado em 30/11/2019.

Wikipedia (2015). Teslacrypt. https://en.wikipedia.org/wiki/TeslaCrypt. Acessado em 01/11/2019.

Wikipedia (2017). Filesystem in userspace (fuse). https://en.wikipedia.org/wiki/Filesystem_in_Userspace. Acessado em 03/12/2019.